LAB REPORT

Submitted by

Arnav Aggarwal(RA2111032010002)
Navdeep Singh Jakhar (RA2111032010030)

Under the Guidance of

Dr. A.Prabhu Chakkaravarthy

Assistant Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

with specialization in Internet of Things

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR - 603203

MAY 2023

COLLEGE OF ENGINEERING &
TECHNOLOGY SRM INSTITUTE OF
SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603 203
Chengalpattu District

BONAFIDE CERTIFICATE

Register No. RA2111032010002, RA2111032010030 Certified to be the bonafide work done by Arnav Aggarwal, Navdeep Singh Jakhar of II Year/IV Sem B. Tech Degree Course in the **Practical Course – 18CSC204J - Design and Analysis of Algorithms** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY,** Kattankulathur during the academic year 2022 – 2023.

SIGNATURE

Faculty In-Charge
Dr. A.Prabhu Chakkaravarthy
Assistant Professor
Department of Networking and Communications
SRM Institute of Science and Technology

SIGNATURE

HEAD OF THE DEPARTMENT
Dr. Annapurani Panaiyappan. K
Professor and Head,
Department of Networking and Communications
SRM Institute of Science and Technology

ABSTRACT

Task scheduling is a well-known problem in computer science, where a set of tasks needs to be scheduled on available resources in order to optimize some performance metric. The problem arises in various contexts, such as in operating systems, cloud computing, and project management.The task scheduling problem involves determining the optimal schedule for a set of tasks with varying processing times, dependencies, and resource requirements. The objective is to minimize the makespan, which is the total time required to complete all tasks.This problem is known to be NP-hard, meaning that there is no known algorithm that can solve it in polynomial time. Therefore, various heuristic and approximation algorithms have been developed to provide practical solutions to this problem.

TABLE OF CONTENTS

PROBLEM DEFINITION

- Task scheduling problem refers to the problem of scheduling a set of tasks on a set of resources such that the overall performance of the system is optimized. The task scheduling problem is a combinatorial optimization problem, where the goal is to minimize the makespan, which is the total time required to complete all tasks.
- The problem can be formulated as follows: given a set of n tasks, each with a processing time pi, and m identical resources, find a schedule that assigns each task to a resource, such that no two tasks are assigned to the same resource at the same time, and the total completion time of all tasks is minimized.
- The task scheduling problem has several variations, including task dependencies, precedence constraints, release dates, deadlines, and resource constraints. These variations add complexity to the problem and make it more challenging to find an optimal solution.
- The task scheduling problem is important in many real-world applications, such as job scheduling in a data center, task scheduling in a distributed computing system, and project scheduling in software development. The problem is known to be NP-hard, meaning that it is unlikely to find an exact algorithm that solves the problem in polynomial time. Therefore, various heuristic and approximation algorithms have been developed to provide practical solutions to this problem.

WHY GREEDY APPROACH IS THE BEST SUITED ALGORITHM TECHNIQUE FOR TASK SCHEDULING PROBLEM?

- ☐ The greedy approach is a commonly used algorithmic technique for solving task scheduling problems because it is simple, easy to implement, and often provides reasonable solutions. The basic idea of the greedy approach is to make locally optimal decisions at each step, with the hope that the combination of these decisions will lead to a globally optimal solution.

- ☐ In the case of task scheduling problems, the greedy approach can be used to assign tasks to resources in a way that minimizes the overall makespan. This can be done by sorting the tasks in decreasing order of their processing time and then assigning each task to the resource with the earliest available finish time.

- ☐ The greedy approach has several advantages for solving task scheduling problems. Firstly, it is computationally efficient and can handle large problem instances. Secondly, it is easy to implement and can be adapted to handle different variations of the problem, such as task dependencies or resource constraints. Finally, the greedy approach often provides near-optimal solutions and has a provable approximation guarantee for certain variants of the task scheduling problem.

ALGORITHM FOR THE PROBLEM

Here is an algorithm for solving the task scheduling problem using the greedy approach:

Input: A set of n tasks with processing times p1, p2, ..., pn and m resources.

Output: A schedule that assigns each task to a resource such that the overall makespan is minimized.

1. Sort the tasks in decreasing order of their processing times.
2. Create an array finish_time[1...m] to store the finish time of each resource, initially set all elements to zero.
3. For each task i from 1 to n, do the following:

   a. Find the resource r with the earliest finish time among all resources.

   b. Assign task i to resource r.

   c. Update the finish time of resource r by adding the processing time of task i.
4. Return the schedule.

The above algorithm works by selecting the task with the largest processing time first and assigning it to the resource with the earliest finish time. This is repeated for each task in decreasing order of processing time until all tasks are assigned to resources. The algorithm ensures that no two tasks are assigned to the same resource at the same time and that the overall makespan is minimized.

PSEUDO CODE

```
procedure greedy_task_scheduling(tasks[1..n], resources[1..m])
    // Sort tasks in decreasing order of processing time
    sort(tasks, descending order of processing time)

    // Initialize finish time of each resource to 0
    finish_time[1..m] = {0}

    // Assign tasks to resources in greedy manner
    for i = 1 to n do
        // Find the resource with earliest finish time
        r = argmin(finish_time)

        // Assign task i to resource r
        assign task i to resource r

        // Update finish time of resource r
        finish_time[r] = finish_time[r] + processing time of task i

    end for

    // Return the schedule
    return schedule
end procedure
```

In this pseudo code, we first sort the tasks in decreasing order of processing time using the sort() function. We then initialize the finish time of each resource to 0 using the finish_time array.

We then loop through each task i and find the resource r with the earliest finish time using the argmin() function. We assign task i to resource r and update the finish time of resource r by adding the processing time of task i to the current finish time.

Finally, we return the schedule, which is the assignment of each task to a resource.

CODE IMPLEMENTATION IN C

```c
#include <stdio.h>
#include <stdlib.h>

// Define a task struct to store task information
typedef struct {
   int id;
   int processing_time;
} task;

// Define a resource struct to store resource information
typedef struct {
   int id;
   int finish_time;
} resource;

// Define a function to compare two tasks based on processing time (for sorting)
int compare_tasks(const void* a, const void* b) {
   task* task_a = (task*)a;
   task* task_b = (task*)b;
   return task_b->processing_time - task_a->processing_time;
}

// Define the main function for task scheduling
int main() {
   int n = 5;  // number of tasks
   int m = 2;  // number of resources

   // Initialize tasks and resources
   task tasks[] = { {1, 5}, {2, 3}, {3, 7}, {4, 2}, {5, 4} };
   resource resources[] = { {1, 0}, {2, 0} };

   // Sort tasks in decreasing order of processing time
   qsort(tasks, n, sizeof(task), compare_tasks);

   // Assign tasks to resources in greedy manner
   for (int i = 0; i < n; i++) {
```

```
        // Find the resource with earliest finish time
        int r = 0;
        for (int j = 1; j < m; j++) {
            if (resources[j].finish_time < resources[r].finish_time) {
                r = j;
            }
        }
        // Assign task i to resource r
        printf("Task %d assigned to resource %d\n", tasks[i].id,
resources[r].id);

        // Update finish time of resource r
        resources[r].finish_time += tasks[i].processing_time;
    }
    return 0;
}
```

OUTPUT :

```
Task 3 assigned to resource 1
Task 1 assigned to resource 2
Task 5 assigned to resource 2
Task 2 assigned to resource 1
Task 4 assigned to resource 2
```

In this implementation, we define a task struct to store task information (i.e. ID and processing time) and a resource struct to store resource information (i.e. ID and finish time). We then define a comparison function for tasks based on processing time, which is used for sorting the tasks in decreasing order.

In the main function, we initialize the tasks and resources, sort the tasks using the qsort() function, and then loop through each task in the sorted order. For each task, we find the resource with the earliest finish time using a simple linear search and assign the task to that resource. We then update the finish time of the assigned resource by adding the processing time of the task.

Finally, we print out the assignment of each task to a resource, which corresponds to the schedule produced by the algorithm.

COMPLEXITY ANALYSIS

Line of Code	Execution Count	Total Cost
int n = 5;	1	1
int n = 2;	1	1
task tasks[] = {{1, 5} , {2, 3} , {3, 7} , {4, 2} , {5,4}};	1	1
resource resources[] = { {1, 0} , {2, 0} };	1	1
qsort(tasks, n, sizeof(rask), compare_tasks);	1	O(n log n)
for (int i = 0; i < n; i++) {	1	1
int r = 0;	n	n
for (int j = 1; j < m; j++) {	n(m-1)	n(m-1)
if (resources[j].finish_time < resources[r].finsih_time) { r = j; };	n(m-1)	n(m-1)
printf("Task %d assigned to resource %d\n", tasks[i].id, resources[r].id);	n	n
resources[r].finish_time += tasks[i].processing_time;	n	n
}	1	1

Therefore, the total cost of the algorithm is:

$$1 + 1 + 1 + 1 + O(n \log n) + 1 + n + n(m-1) + n(m-1) + n + n + 1$$

$$= O(nm) + O(n \log n) + O(n(m-1))$$

$$= O(nm \log n)$$

Thus, the time complexity of the algorithm is **O(nm log n)**. This means that the running time of the algorithm grows logarithmically with the size of the input, which is relatively efficient for small and medium-sized inputs. However, for very large inputs, the algorithm may become slow due to its dependence on the number of tasks and resources.

CONCLUSION

In conclusion, the task scheduling problem is a fundamental problem in computer science that involves assigning tasks to resources in a way that minimizes the overall completion time. The greedy approach is a commonly used algorithm technique for solving this problem efficiently.

The greedy approach for the task scheduling problem involves sorting the tasks in non-decreasing order of their processing time and then iteratively assigning each task to the resource that has the earliest available finish time. This algorithm has a time complexity of O(nm log n), where n is the number of tasks and m is the number of resources.

While the greedy approach does not always guarantee an optimal solution, it is often a good heuristic for solving the task scheduling problem efficiently in practice. The simplicity and efficiency of the algorithm make it a popular choice for many real-world scheduling applications. However, it is important to note that the performance of the algorithm depends on the input data and the problem constraints, and it may not always be the best approach for every scheduling problem.

REFERENCES

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). The MIT Press. Chapter 16: Greedy Algorithms.

[2] Pinedo, M. (2012). Scheduling: Theory, algorithms, and systems (4th ed.). Springer.

[3] Kaul, M., & Gupta, A. (2017). A comparative study of heuristic algorithms for task scheduling problem. International Journal of Engineering Science and Computing, 7(12), 17130-17134.

[4] Bai, Y., & Yang, L. (2016). A new greedy algorithm for task scheduling in cloud computing environment. Journal of Computational and Theoretical Nanoscience, 13(9), 6245-6251.

[5] Fong, S., & Siu, T. (2005). A greedy algorithm for scheduling tasks with resource constraints. Journal of Scheduling, 8(1), 43-54.

[6] Zhong, Y., Wang, Q., & Xie, H. (2017). An improved greedy algorithm for task scheduling in cloud computing. International Journal of Computer Science and Network Security, 17(6), 102-107.